15-744: Computer Networking

Queuing
Queuing

- Fair Queuing
- Core-stateless Fair queuing
- Assigned reading
  - [DKS90] Analysis and Simulation of a Fair Queueing Algorithm, Internetworking: Research and Experience
  - [XCP] Congestion Control for High Bandwidth-Delay Product Networks

- Optional
  - [SSZ98] Core-Stateless Fair Queueing: Achieving Approximately Fair Allocations in High Speed Networks
Overview

• TCP and queues
• Queuing disciplines
• RED
• Fair-queuing
• Core-stateless FQ
• XCP
Queuing Disciplines

- Each router must implement some queuing discipline
- Queuing allocates both bandwidth and buffer space:
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- Queuing also affects latency
Packet Drop Dimensions

Aggregation

Per-connection state

Class-based queuing

Drop position

Single class

Head

Tail

Random location

Early drop

Overflow drop
Typical Internet Queuing

- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
- FIFO (first-in-first-out)
  - Implies single class of traffic
- Drop-tail
  - Arriving packets get dropped when queue is full regardless of flow or importance
- Important distinction:
  - FIFO: scheduling discipline
  - Drop-tail: drop policy
FIFO + Drop-tail Problems

- Leaves responsibility of congestion control to edges (e.g., TCP)
- Does not separate between different flows
- No policing: send more packets → get more service
- Synchronization: end hosts react to same events
Active Queue Management

• Design active router queue management to aid congestion control

• Why?
  • Routers can distinguish between propagation and persistent queuing delays
  • Routers can decide on transient congestion, based on workload
Active Queue Designs

- Modify both router and hosts
  - DECbit – congestion bit in packet header
- Modify router, hosts use TCP
  - Fair queuing
    - Per-connection buffer allocation
  - RED (Random Early Detection)
    - Drop packet or set bit in packet header as soon as congestion is starting
Overview

• TCP and queues
• Queuing disciplines
• RED
• Fair-queuing
• Core-stateless FQ
• XCP
Internet Problems

• Full queues
  • Routers are forced to have large queues to maintain high utilizations
  • TCP detects congestion from loss
    • Forces network to have long standing queues in steady-state

• Lock-out problem
  • Drop-tail routers treat bursty traffic poorly
  • Traffic gets synchronized easily → allows a few flows to monopolize the queue space
Design Objectives

• Keep throughput high and delay low
• Accommodate bursts
• Queue size should reflect ability to accept bursts rather than steady-state queuing
• Improve TCP performance with minimal hardware changes
Lock-out Problem

• Random drop
  • Packet arriving when queue is full causes some random packet to be dropped

• Drop front
  • On full queue, drop packet at head of queue

• Random drop and drop front solve the lock-out problem but not the full-queues problem
Full Queues Problem

- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion
  - Example: early random drop (ERD):
    - If qlen > drop level, drop each new packet with fixed probability $p$
    - Does not control misbehaving users
Random Early Detection (RED)

- Detect incipient congestion, allow bursts
- Keep power (throughput/delay) high
  - Keep average queue size low
  - Assume hosts respond to lost packets
- Avoid window synchronization
  - Randomly mark packets
- Avoid bias against bursty traffic
- Some protection against ill-behaved users
RED Algorithm

• Maintain running average of queue length
• If $\text{avgq} < \text{min}_{th}$ do nothing
  • Low queuing, send packets through
• If $\text{avgq} > \text{max}_{th}$, drop packet
  • Protection from misbehaving sources
• Else mark packet in a manner proportional to queue length
  • Notify sources of incipient congestion
RED Operation

Max thresh

Min thresh

Average Queue Length

P(drop)

1.0

max_p

min_th

max_th

Avg queue length
Queue Estimation

• Standard EWMA: \( \text{avgq} = (1-w_q) \text{avgq} + w_q \text{qlen} \)
  • Special fix for idle periods – why?

• Upper bound on \( w_q \) depends on \( \min_{th} \)
  • Want to ignore transient congestion
  • Can calculate the queue average if a burst arrives
    • Set \( w_q \) such that certain burst size does not exceed \( \min_{th} \)

• Lower bound on \( w_q \) to detect congestion relatively quickly

• Typical \( w_q = 0.002 \)
Thresholds

- \( \text{min}_{\text{th}} \) determined by the utilization requirement
  - Tradeoff between queuing delay and utilization
- Relationship between \( \text{max}_{\text{th}} \) and \( \text{min}_{\text{th}} \)
  - Want to ensure that feedback has enough time to make difference in load
  - Depends on average queue increase in one RTT
  - Paper suggest ratio of 2
    - Current rule of thumb is factor of 3
Packet Marking

- $\text{max}_p$ is reflective of typical loss rates
- Paper uses 0.02
  - 0.1 is more realistic value
- If network needs marking of 20-30% then need to buy a better link!
- Gentle variant of RED (recommended)
  - Vary drop rate from $\text{max}_p$ to 1 as the avgq varies from $\text{max}_\text{th}$ to $2*\text{max}_\text{th}$
  - More robust to setting of $\text{max}_\text{th}$ and $\text{max}_p$
Extending RED for Flow Isolation

• Problem: what to do with non-cooperative flows?
• Fair queuing achieves isolation using per-flow state – expensive at backbone routers
  • How can we isolate unresponsive flows without per-flow state?
• RED penalty box
  • Monitor history for packet drops, identify flows that use disproportionate bandwidth
  • Isolate and punish those flows
Overview

• TCP and queues
• Queuing disciplines
• RED
• Fair-queuing
• Core-stateless FQ
• XCP
Fairness Goals

• Allocate resources fairly
• Isolate ill-behaved users
  • Router does not send explicit feedback to source
  • Still needs e2e congestion control
• Still achieve statistical muxing
  • One flow can fill entire pipe if no contenders
  • Work conserving → scheduler never idles link if it has a packet
What is Fairness?

• At what granularity?
  • Flows, connections, domains?

• What if users have different RTTs/links/etc.
  • Should it share a link fairly or be TCP fair?

• Maximize fairness index?
  • Fairness = \( \frac{(\sum x_i)^2}{n(\sum x_i^2)} \)  \( 0 < \text{fairness} < 1 \)

• Basically a tough question to answer – typically design mechanisms instead of policy
  • User = arbitrary granularity
Max-min Fairness

- Allocate user with “small” demand what it wants, evenly divide unused resources to “big” users
- Formally:
  - Resources allocated in terms of increasing demand
  - No source gets resource share larger than its demand
  - Sources with unsatisfied demands get equal share of resource
Max-min Fairness Example

• Assume sources 1..n, with resource demands $X_1..X_n$ in ascending order

• Assume channel capacity $C$.
  • Give $C/n$ to $X_1$; if this is more than node 1 wants, divide excess $(C/n - X_1)$ to other sources: each gets $C/n + (C/n - X_1)/(n-1)$
  • If this is larger than what $X_2$ wants, repeat process
Implementing max-min Fairness

• Generalized processor sharing
  • Fluid fairness
  • Bitwise round robin among all queues
• Why not simple round robin?
  • Variable packet length $\rightarrow$ can get more service by sending bigger packets
  • Unfair instantaneous service rate
    • What if arrive just before/after packet departs?
Bit-by-bit RR

• Single flow: clock ticks when a bit is transmitted. For packet i:
  • $P_i$ = length, $A_i$ = arrival time, $S_i$ = begin transmit time, $F_i$ = finish transmit time
  • $F_i = S_i + P_i = \max (F_{i-1}, A_i) + P_i$

• Multiple flows: clock ticks when a bit from all active flows is transmitted $\rightarrow$ round number
  • Can calculate $F_i$ for each packet if number of flows is known at all times
    • This can be complicated
Bit-by-bit RR Illustration

- Not feasible to interleave bits on real networks
  - FQ simulates bit-by-bit RR
Fair Queuing

- Mapping bit-by-bit schedule onto packet transmission schedule
- Transmit packet with the lowest $F_i$ at any given time
  - How do you compute $F_i$?
Bit-by-bit RR Example

Flow 1
F=8

F=10

Flow 2

Output

Flow 1 (arriving)

Flow 2 transmitting

Output

F=10

F=2

Cannot preempt packet currently being transmitted
Fair Queuing Tradeoffs

• FQ can control congestion by monitoring flows
  • Non-adaptive flows can still be a problem – why?

• Complex state
  • Must keep queue per flow
    • Hard in routers with many flows (e.g., backbone routers)
    • Flow aggregation is a possibility (e.g. do fairness per domain)

• Complex computation
  • Classification into flows may be hard
  • Must keep queues sorted by finish times
  • Finish times change whenever the flow count changes
Overview

• TCP and queues
• Queuing disciplines
• RED
• Fair-queuing
• Core-stateless FQ
• XCP
Core-Stateless Fair Queuing

- Key problem with FQ is core routers
  - Must maintain state for 1000’s of flows
  - Must update state at Gbps line speeds
- CSFQ (Core-Stateless FQ) objectives
  - Edge routers should do complex tasks since they have fewer flows
  - Core routers can do simple tasks
    - No per-flow state/processing → this means that core routers can only decide on dropping packets not on order of processing
    - Can only provide max-min bandwidth fairness not delay allocation
Core-Stateless Fair Queuing

• Edge routers keep state about flows and do computation when packet arrives

• DPS (Dynamic Packet State)
  • Edge routers label packets with the result of state lookup and computation

• Core routers use DPS and local measurements to control processing of packets
Edge Router Behavior

• Monitor each flow $i$ to measure its arrival rate ($r_i$)
  • EWMA of rate
  • Non-constant EWMA constant
    • $e^{-T/K}$ where $T =$ current interarrival, $K =$ constant
    • Helps adapt to different packet sizes and arrival patterns
• Rate is attached to each packet
Core Router Behavior

• Keep track of fair share rate $\alpha$
  • Increasing $\alpha$ does not increase load ($F$) by $N \cdot \alpha$
  • $F(\alpha) = \sum_i \min(r_i, \alpha) \rightarrow$ what does this look like?

• Periodically update $\alpha$

• Keep track of current arrival rate
  • Only update $\alpha$ if entire period was congested or uncongested

• Drop probability for packet $\rightarrow$ max($1 - \alpha/r, 0$)
F vs. Alpha

C [linked capacity]
Estimating Fair Share

• Need \( F(\alpha) = \text{capacity} = C \)
  • Can’t keep map of \( F(\alpha) \) values → would require per flow state
  • Since \( F(\alpha) \) is concave, piecewise-linear
    • \( F(0) = 0 \) and \( F(\alpha) = \text{current accepted rate} = F_c \)
    • \( F(\alpha) = F_c / \alpha \)
    • \( F(\alpha_{\text{new}}) = C \rightarrow \alpha_{\text{new}} = \alpha_{\text{old}} \times C/F_c \)

• What if a mistake was made?
  • Forced into dropping packets due to buffer capacity
  • When queue overflows \( \alpha \) is decreased slightly
Overview

• TCP and queues
• Queuing disciplines
• RED
• Fair-queuing
• Core-stateless FQ
• XCP
How does XCP Work?

Round Trip Time

Congestion Window

Feedback = + 0.1 packet

Congestion Header
How does XCP Work?
How does XCP Work?

XCP extends ECN and CSFQ

Routers compute feedback without any per-flow state
How Does an XCP Router Compute the Feedback?

**Congestion Controller**

- Goal: Divides $D$ between flows to converge to fairness
- Looks at a flow's state in Congestion Header
- Algorithm:
  - If $D > 0$, divide $D$ equally between flows
  - If $D < 0$, divide $D$ between flows proportionally to their current rates

**Fairness Controller**

- Goal: Matches input traffic to link capacity & drains the queue
- Looks at aggregate traffic & queue
- Algorithm:
  - Aggregate traffic changes by $\Delta$
  - $\Delta \sim$ Spare Bandwidth
  - $\Delta \sim -$ Queue Size
  - So, $\Delta = \alpha d_{avg} \text{Spare} - \beta \text{Queue}$

**MIMD**

**AIMD**
Getting the devil out of the details …

**Congestion Controller**

\[ \Delta = \alpha d_{avg} \text{Spare} - \beta \text{Queue} \]

**Theorem:** System converges to optimal utilization (i.e., stable) for any link bandwidth, delay, number of sources if:

\[ 0 < \alpha < \frac{\pi}{4\sqrt{2}} \quad \text{and} \quad \beta = \alpha^2 \sqrt{2} \]

**Fairness Controller**

**Algorithm:**

- If \( \Delta > 0 \) \( \Rightarrow \) Divide \( \Delta \) equally between flows
- If \( \Delta < 0 \) \( \Rightarrow \) Divide \( \Delta \) between flows proportionally to their current rates

Need to estimate number of flows \( N \)

\[
N = \sum_{\text{pkts in } T} \frac{1}{T \times (Cwnd_{pkt} / RTT_{pkt})}
\]

\( RTT_{pkt} \): Round Trip Time in header

**No Parameter Tuning**

**No Per-Flow State**
Discussion

- RED
  - Parameter settings
- RED vs. FQ
  - How much do we need per flow tracking? At what cost?
- FQ vs. XCP/CSFQ
  - Is coarse-grained fairness sufficient?
  - Misbehaving routers/trusting the edge
  - Deployment (and incentives)
  - How painful is FQ
- XCP vs CSFQ
  - What are the key differences
- Granularity of fairness
  - Mechanism vs. policy \(\rightarrow\) will see this in QoS
Important Lessons

• How does TCP implement AIMD?
  • Sliding window, slow start & ack clocking
  • How to maintain ack clocking during loss recovery → fast recovery

• How does TCP fully utilize a link?
  • Role of router buffers

• TCP alternatives
  • TCP being used in new/unexpected ways
  • Key changes needed
Lessons

- Fairness and isolation in routers
  - Why is this hard?
  - What does it achieve – e.g. do we still need congestion control?

- Routers
  - FIFO, drop-tail interacts poorly with TCP
  - Various schemes to desynchronize flows and control loss rate (e.g. RED)

- Fair-queuing
  - Clean resource allocation to flows
  - Complex packet classification and scheduling

- Core-stateless FQ & XCP
  - Coarse-grain fairness
  - Carrying packet state can reduce complexity
Next: Data center topology/routing

• What do we make Ethernet scale to data center sizes?

• Assigned reading
  • Portland (read fully)
  • Google (skim)